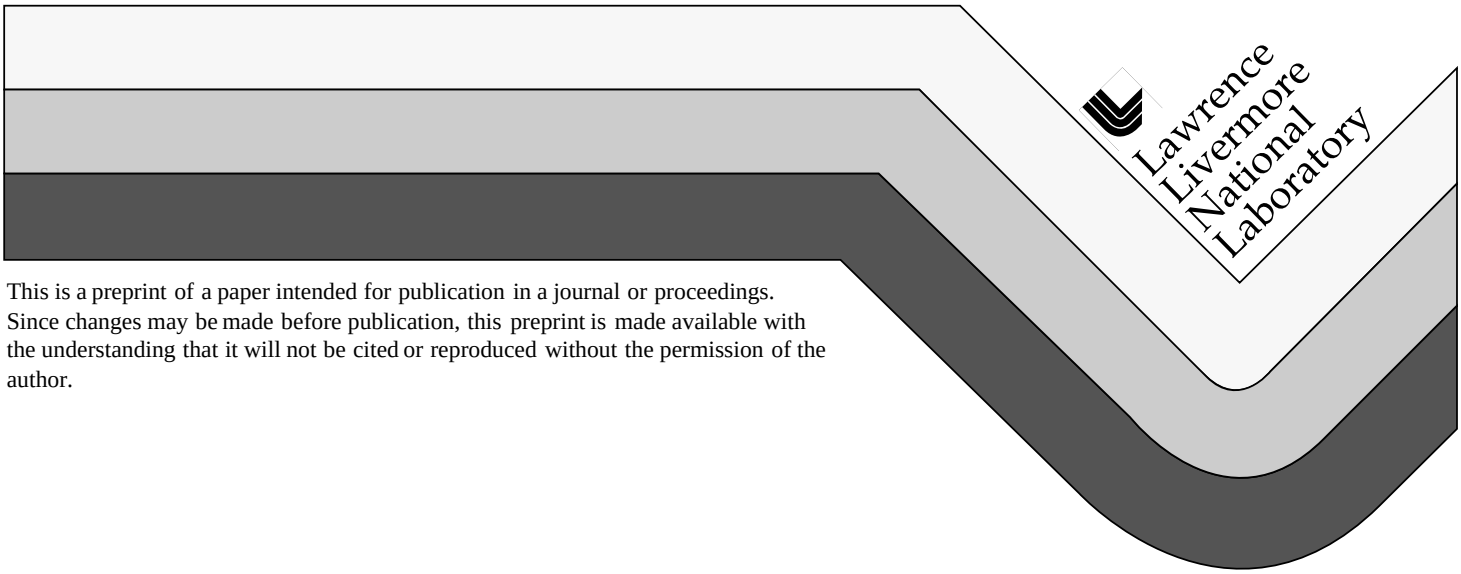
UCRL-JC-132743
PREPRINT

The Kull IMC Package

N. A. Gentile, N. Keen, J.Rathkopf

This paper was prepared for submittal to the
1998 Nuclear Explosives Development Conference
Las Vegas, NV
October 25-30, 1998

October 1, 1998



The Kull IMC Package

Nicholas A. Gentile, Noel Keen, Jim Rathkopf
Lawrence Livermore National Laboratory

We describe the Kull IMC package, and Implicit Monte Carlo Program written for use in A and X division radiation hydro codes. The Kull IMC has been extensively tested. Written in C++ and using genericity via the template feature to allow easy integration into different codes, the Kull IMC currently runs coupled radiation hydrodynamic problems in 2 different 3D codes. A stand-alone version also exists, which has been parallelized with mesh replication. This version has been run on up to 384 processors on ASCI Blue Pacific.

Keywords: Monte Carlo

Introduction

Developing scientific codes has usually been done without much emphasis on some of the factors that the commercial software industry regards as very important goals: code reuse, ease of use, avoiding duplication of effort. The most important reason for this is that developing algorithms to solve scientific problems is so difficult that other considerations are of necessity neglected. (A second reason is probably that we can't sell the codes.)

But paying more attention to some of these aspects of software development could have benefits for writing the next generation of scientific codes, and so using tools and concepts that have proven valuable in commercial applications could be profitable. The Kull project is attempting to use Object-Oriented code design, implemented with C++, to make code development easier, make debugging less painful, and allow for different pieces of the code to be used by other codes. The current status of the Kull IMC package demonstrates some success in achieving this.

In the following, we will give an overview of the Kull IMC packages current capabilities, discuss how using C++ has been beneficial, show results of various applications of the code, and discuss efforts to parallelize the code.

Current Capabilities of the Kull IMC package

The Kull IMC package can run radiation hydrodynamic problems in 2 different 3D codes. The Kull hydrodynamics code has Lagrangean and ALE modes, both of which run on meshes consisting of arbitrary polyhedra. This code is written in C++, and has an steerable user interface written in Python. Kull IMC, when run in conjunction with the Kull hydro, is also controlled from Python. The Hydra code is a 3D ALE code written in C. It has meshes consisting of linked KLM blocks. The hexes in these blocks can have degenerate nodes

Kull IMC also exists in a stand-alone version that can run on 1, 2, and 3D orthogonal cartesian

meshes and 2D RZ meshes, as well as a 3D arbitrary polyhedral mesh. This version can be run from Python or as a standard executable.

The stand-alone version has been parallelized using mesh replication, i.e., a copy of the mesh exists on each processor. This version has run on 32 processors of ASCI Blue Mountain, and on up to 384 processors on ASCI Blue Pacific.

Figure 1 shows the results of an analytic Marshak wave test problem (Su and Olson 1996) run on both an orthogonal cartesian mesh and on a mesh of the same dimensions in which each hex has been cut up into 24 tetrahedra. Results on both meshes agree with the analytic result obtained by diffusion theory except at the front of the wave. Here, the superluminal component of the diffusion theory results causes a slight overestimate of the position of the Marshak wave. The IMC results correctly place the foot of the Marshak wave at ct away from the source.

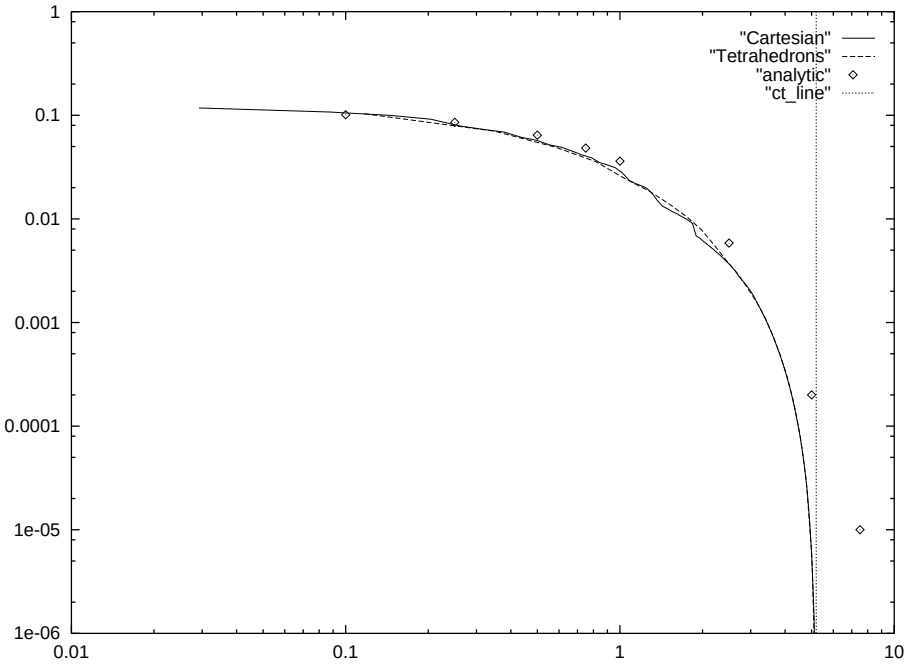


Figure 1. Kull IMC results for a Marshak wave test problem on both structured and unstructured meshes.

Figure 2 shows results for the Top Hat problem of Graziani. This figure shows temperatures for 5 points in the problem as a function of time. Results for Kull IMC, the Teton Sn package, and another IMC code are shown. All three codes agree very well.

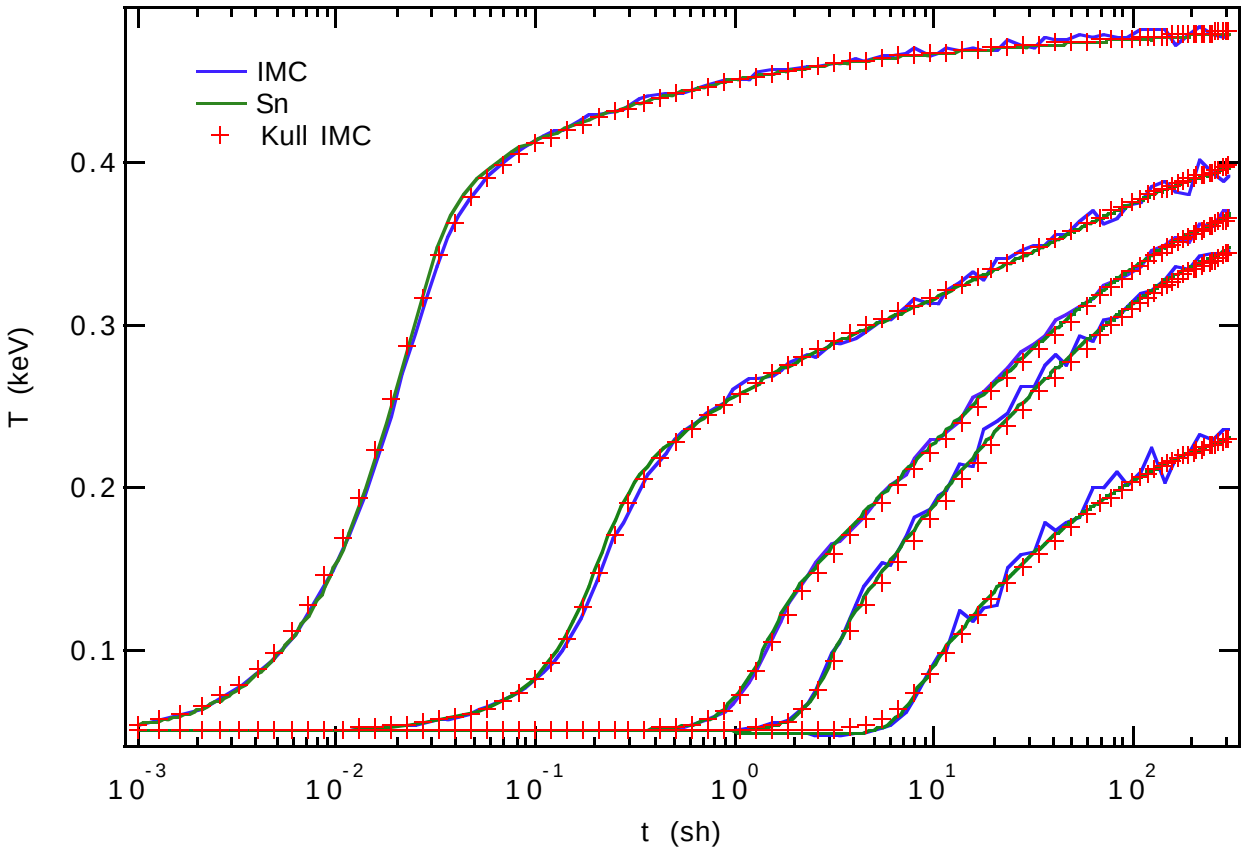


Figure 2. Temperature vs. time for 5 points in the Graziani Top Hat problem.

Figures 3 and 4 demonstrates the IMC running with the kull hydrodynamics package. The mesh (provided by PMESH) is an unstructured grid with about 5000 zones representing a NIF hohlraum. Temperature sources are applied to the outside in the approximate locations of the spots that the NIF lasers will impact the hohlraum. These heated zones radiate and cause the outside of the target to ablate. This compresses the inside of the target. Figure 3 is a temperature plot, showing the heated regions on the outside of the mesh. Figure 4 is a density plot of the target, showing a rarefaction on the outside as the heated exterior ablates away, and also showing compression of the interior.

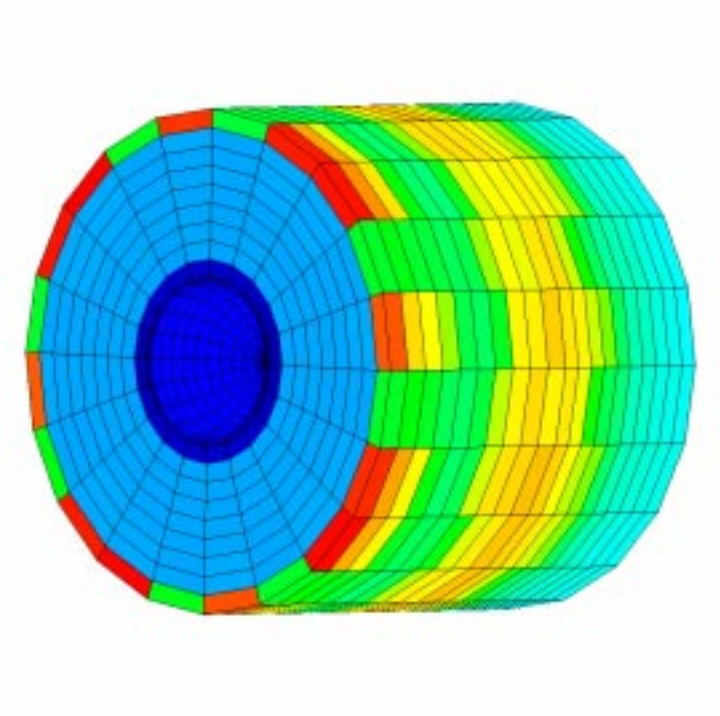


Figure 3. Temperature plot of hohlraum surface.

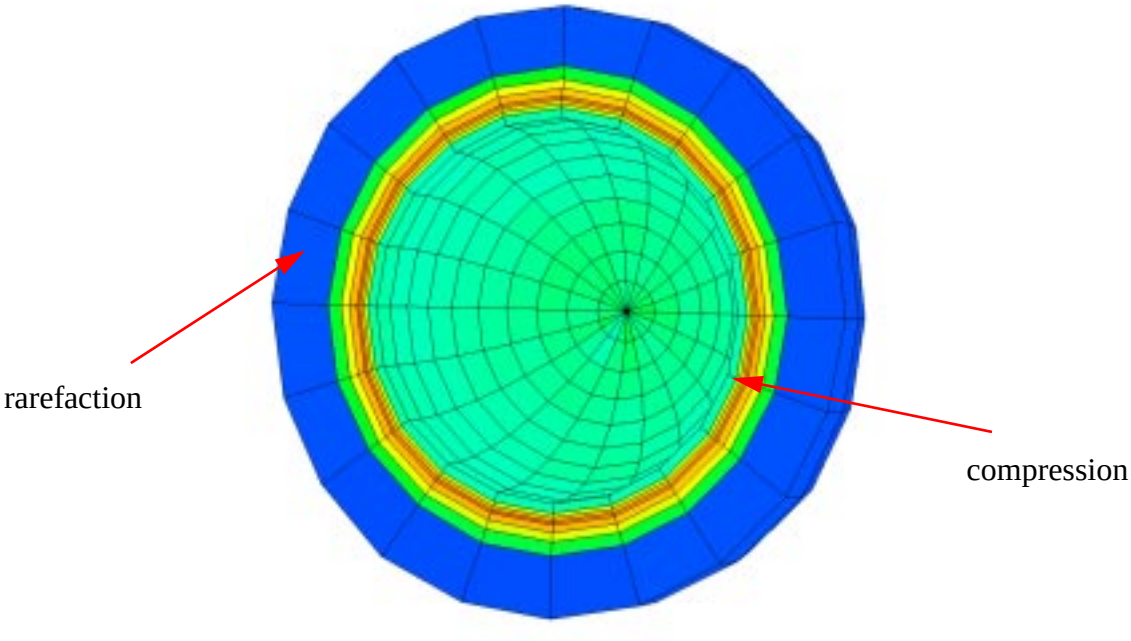


Figure 4. Density plot of target in NIF hohlraum, showing rarefaction and compression waves.

Figure 6 shows a radiation hydrodynamics problem run with Hydra. In this problem, a corner of a cube is heated, which launches a shock into the material. The ALE hydro allows the mesh to distort but does not allow it to tangle.

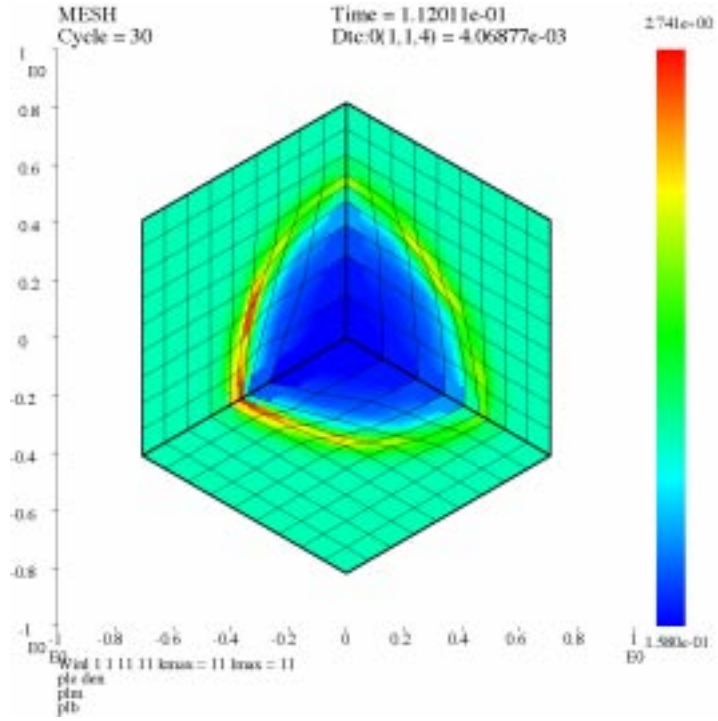


Figure 6. Density plot of a radiation driven shock in the Hydra code.

Use of C++ in the Kull IMC Package

Using Object-Oriented C++ has helped immensely in allowing the Kull IMC package to run on different mesh types with different dimensionality and in coupling the Kull IMC package to different hydrodynamics codes. There are two main features of C++ that make this easier than it would be in a non-object-oriented language. The first is the ability to use objects themselves. The second is templating.

Objects are collections of functions and data that collectively provide some kind of behavior. Objects are a very convenient way of providing an interface between the IMC and different aspects of the code we are trying to link to. For example, codes can have very different meshes, using very different data structures to represent very different ideas about the types of allowable zones, faces, etc. Whatever the features of the meshes, the IMC needs the same kind of things from each one. For example, the IMC needs to have a distance to boundary function to track particles. For each mesh type, we can create an object that has a d_boundary(x, y, z) function that is implemented in terms of the mesh. For the Kull mesh, we write d_boundary in terms of the complex structures needed to describe the arbitrary polyhedral grid. For the hydra mesh, we can write d_boundary in terms of the C pointers used by the hydra code. In either case, the IMC routines

call the d_boundary function of the Mesh object and so are written to be totally independent of the underlying grid.

Templates help in two ways – by providing compile time polymorphism and by allowing the IMC code to use type information provided by the mesh.

Polymorphism means that many similar objects can be used by a piece of code in the same way without having to specify each time which particular object is meant. That is, we do not have to put if tests in the code every time an object is used. In our case, we want to use different meshes without having to specify each time what the particular mesh type is. The most common way of having polymorphism in C++ is to use virtual inheritance. However, this entails some run-time costs; about the same as if we had used if statements to distinguish between mesh types. Templates allow us to specify at compile time which mesh type we want, and allows us to avoid the run-time cost.

Besides providing us with cheaper polymorphic behavior, templates allow us to get information on data types from the mesh. Quantities like density, opacity, etc. need to be stored differently on different types of meshes. For example, density may be a simple array on a 1D mesh, and may be a more complicated structure on a mesh composed of blocks merged together. In order to write the IMC code in a manner that is independent of the data type, we use templates to base the type on the mesh type. Rather than having to commit to a certain data type, we use the type MeshType::Field, which the complier resolves at compile time to be the appropriate entity to hold data on that mesh. Using templates this way is very important in allowing the Kull IMC package to be used with different types of meshes.

Example of Parallel Performance

The parallel version of the Kull IMC package shows good performance on many processors. As an example of a parallel run, we show results for a Nova hohlraum. The Nova laser has 10 beams, which causes 5 hot spots on each side of a hohlraum. Because current codes are only 2D, they model the hot spots as a ring. This causes the illumination of the target to be smoother in the simulation than it is in reality. A 3D calculation, done in parallel on 32 processors of ASCI Blue Mountain, shows that the target illumination has a 5-fold symmetry, which 2D calculations cannot model.

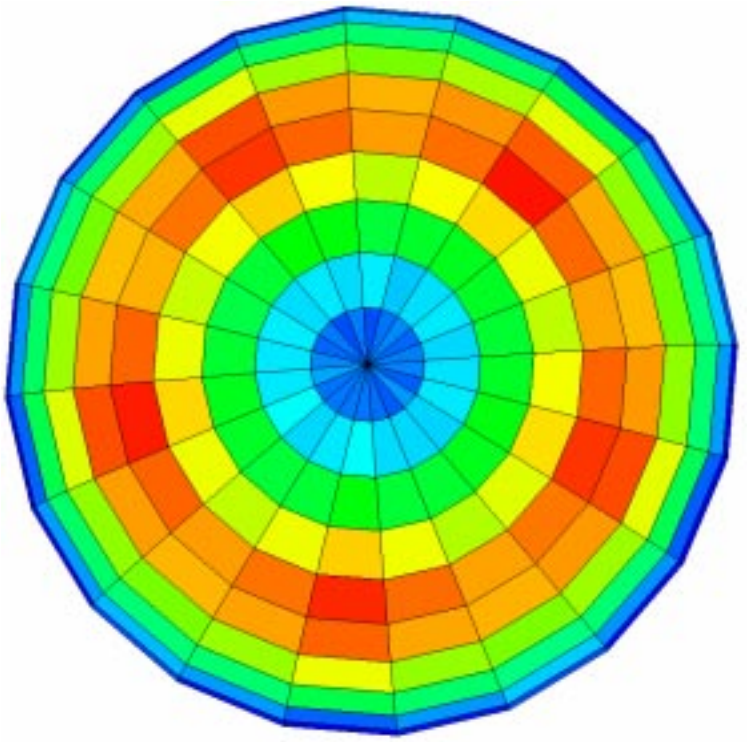
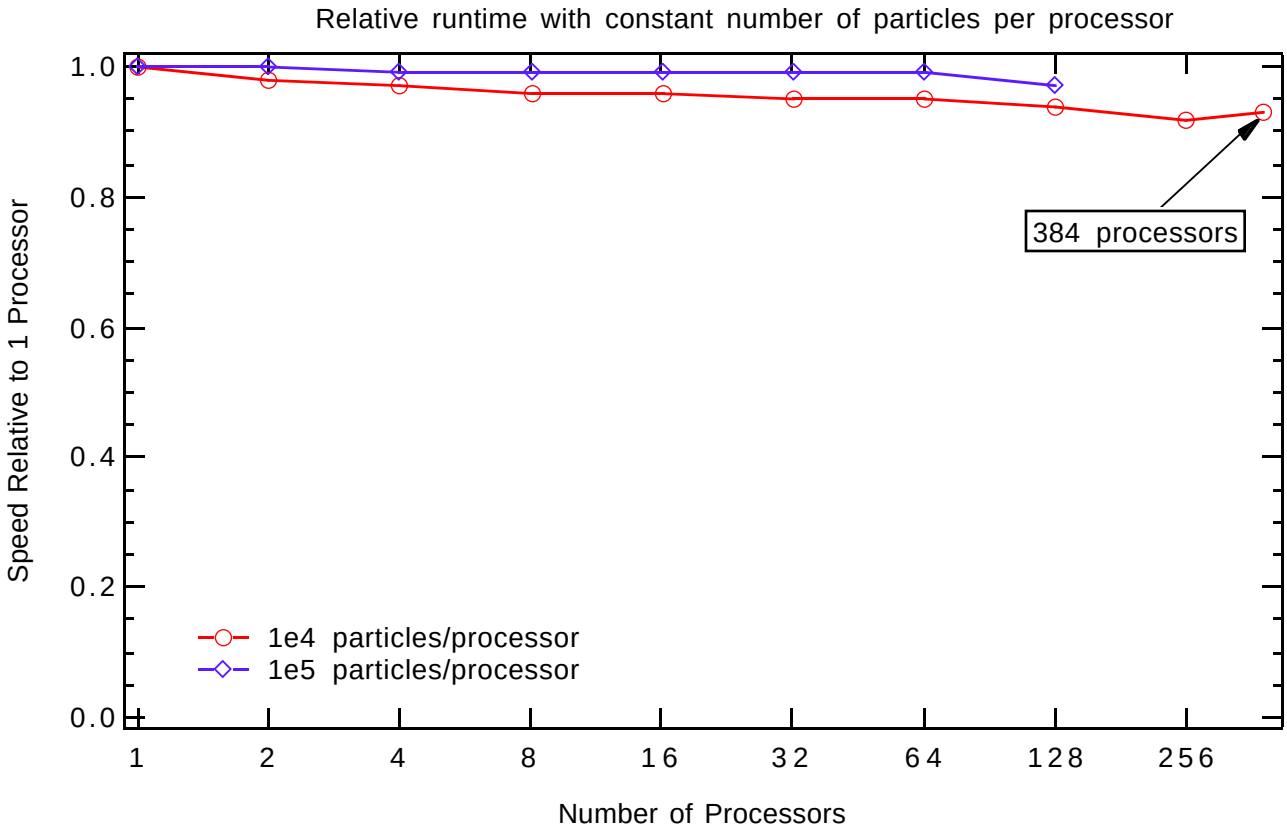


Figure 7. Energy deposition on surface of ablator shows 5-fold symmetry.

As we increase the number of processors, communication time becomes a larger and larger percentage of total runtime. This causes performance to degrade. For the Kull IMC package, we have observed only an 8% drop-off in performance while running on 384 processors of ASCI Blue Pacific. Figure 8 shows the speed relative to 1 processor as we replicate the problem on more and more processors. Ideally, replicating the problem would not cause any slowdown. But the need to communicate causes some degradation in performance. As figure 8 shows, this amount is small.



Conclusion

The Kull IMC package has been tested by comparison to annulate results a, to other IMC codes, and to other radiation transport methods. It currently is capable of running radiation hydrodynamics problems in 2 codes, Kull and Hydra. The stand-alone mesh replicated parallel version has run on up to 384 processors on ASCI Blue Pacific, and shows good speed-ups. C++ and Object-oriented techniques have proven very useful in merging the package with other codes.

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.

References

Su and Olson 1996 Journal of Quantitative Spectroscopy and Radiative Transfer 56, 337-351